

Interception and Analysis of Malicious Traffic Based on NDIS IM Driver

LEE LING CHUAN

Senior Malware Analyst

Malaysia Computer Emergency Response Team (MyCERT)

8-July-2010

About Me

- Lee Ling Chuan
- Senior Malware Analyst at Malware Research Center in Malaysia Computer Emergency Response Team (MyCERT), CyberSecurity Malaysia

Introduction of NDIS

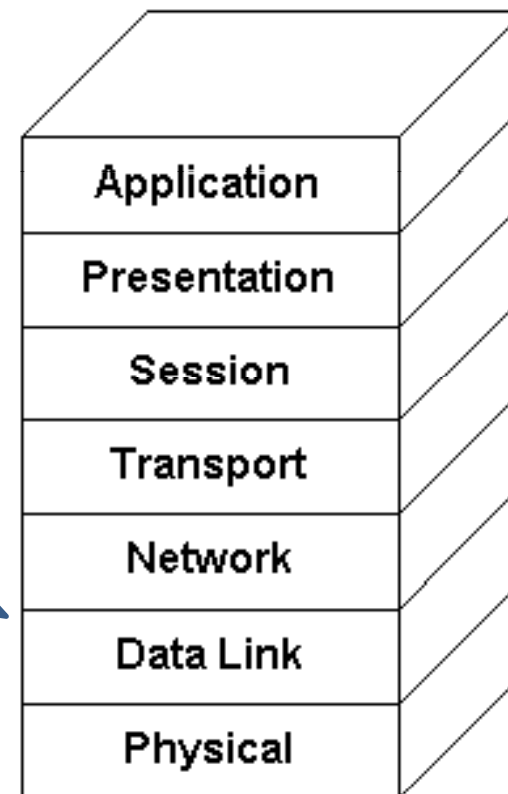
Introduction

- Network Driver Interface Specification (NDIS)
 - Is an application programming interface (API) for network interface cards (NICs).
 - Is a specification for network driver architecture that allows transport protocols such as TCP/IP, Native ATM, IPX and NetBEUI to communicate with an underlying network adapter or other hardware device.

Introduction

The OSI Reference Model

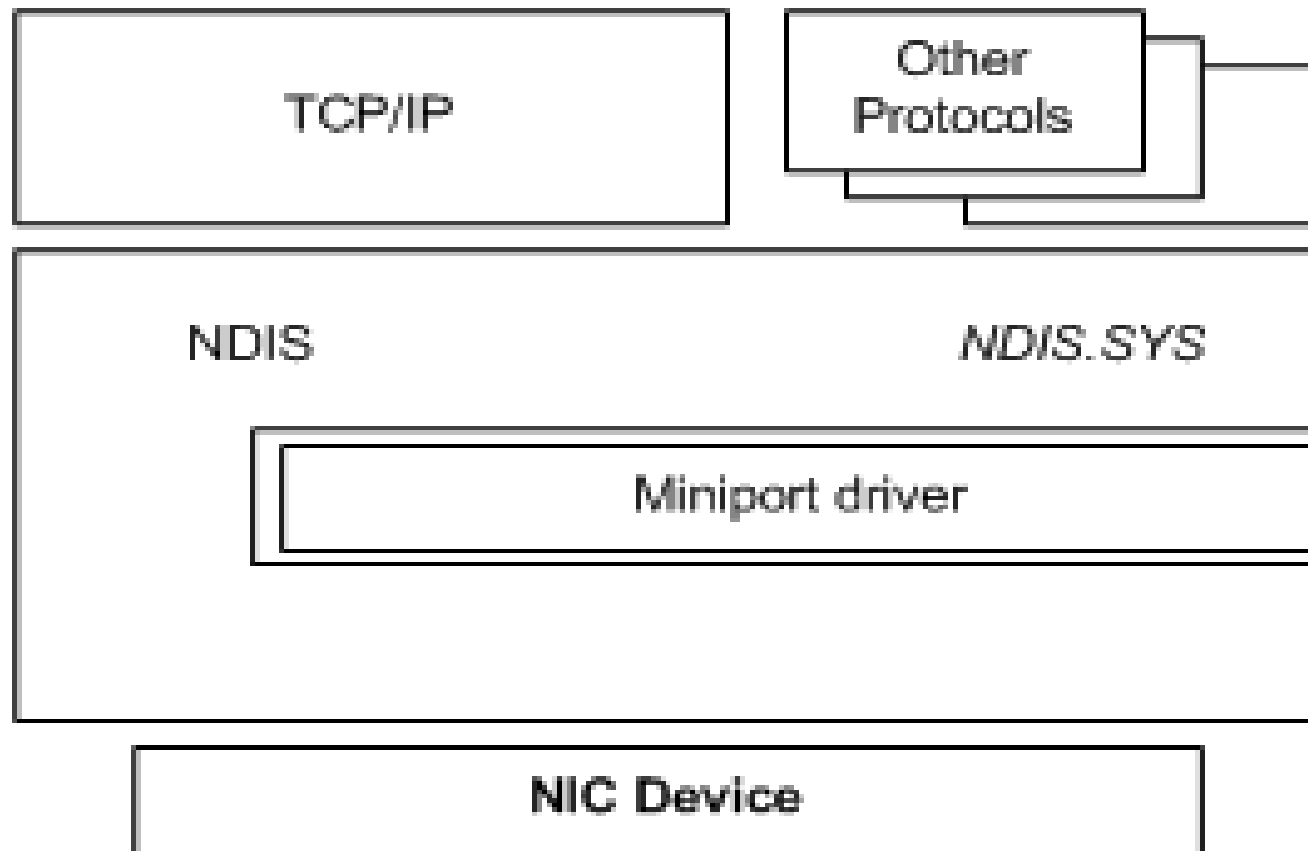
NDIS is a layer of abstraction that exists between the network protocol driver (network layer) and the network card driver (at the data link layer)



Physical Medium

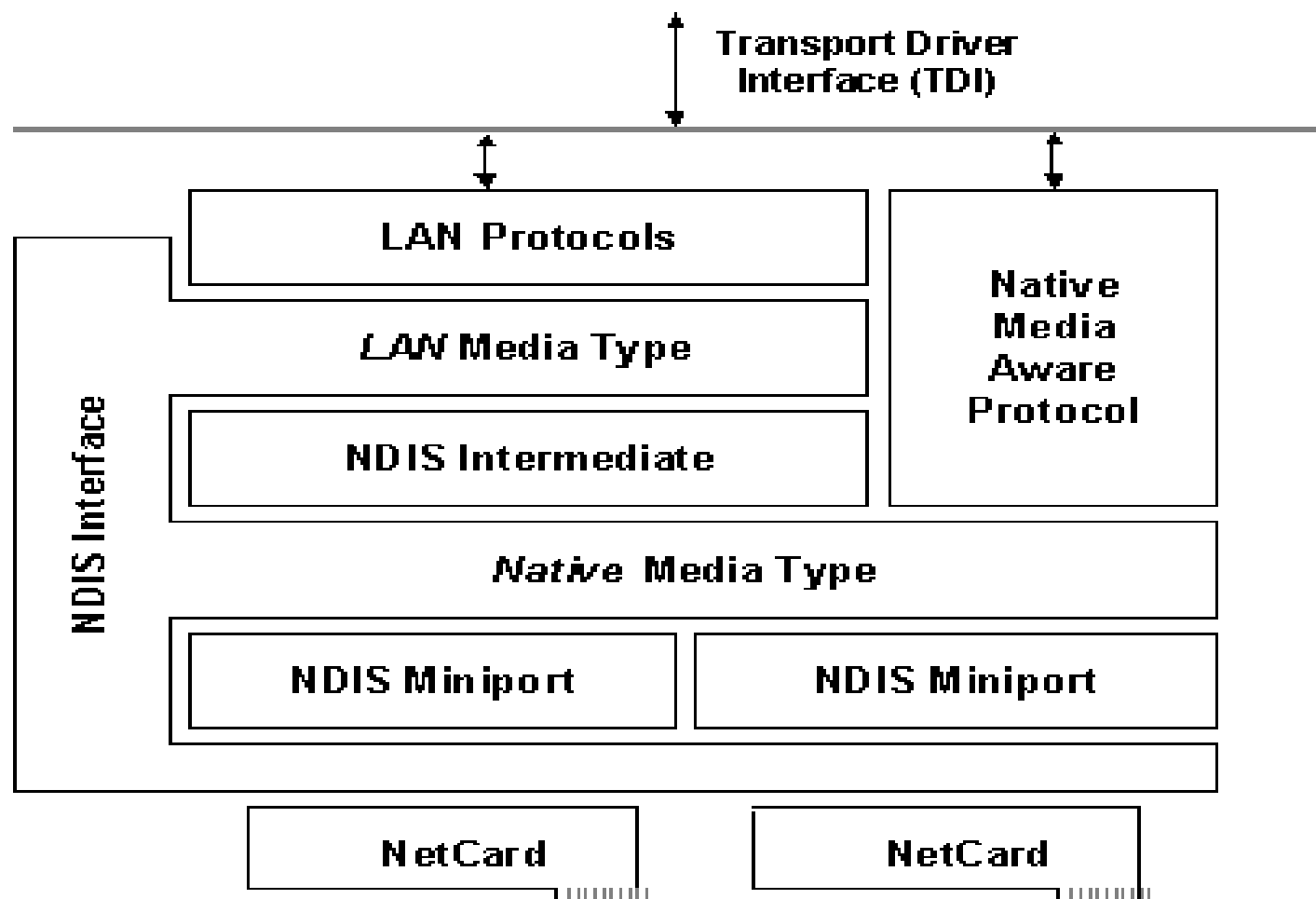
(Reference: MSDN Microsoft)

The Structure of NDIS



Reference: MSDN Microsoft

The structure of NDIS

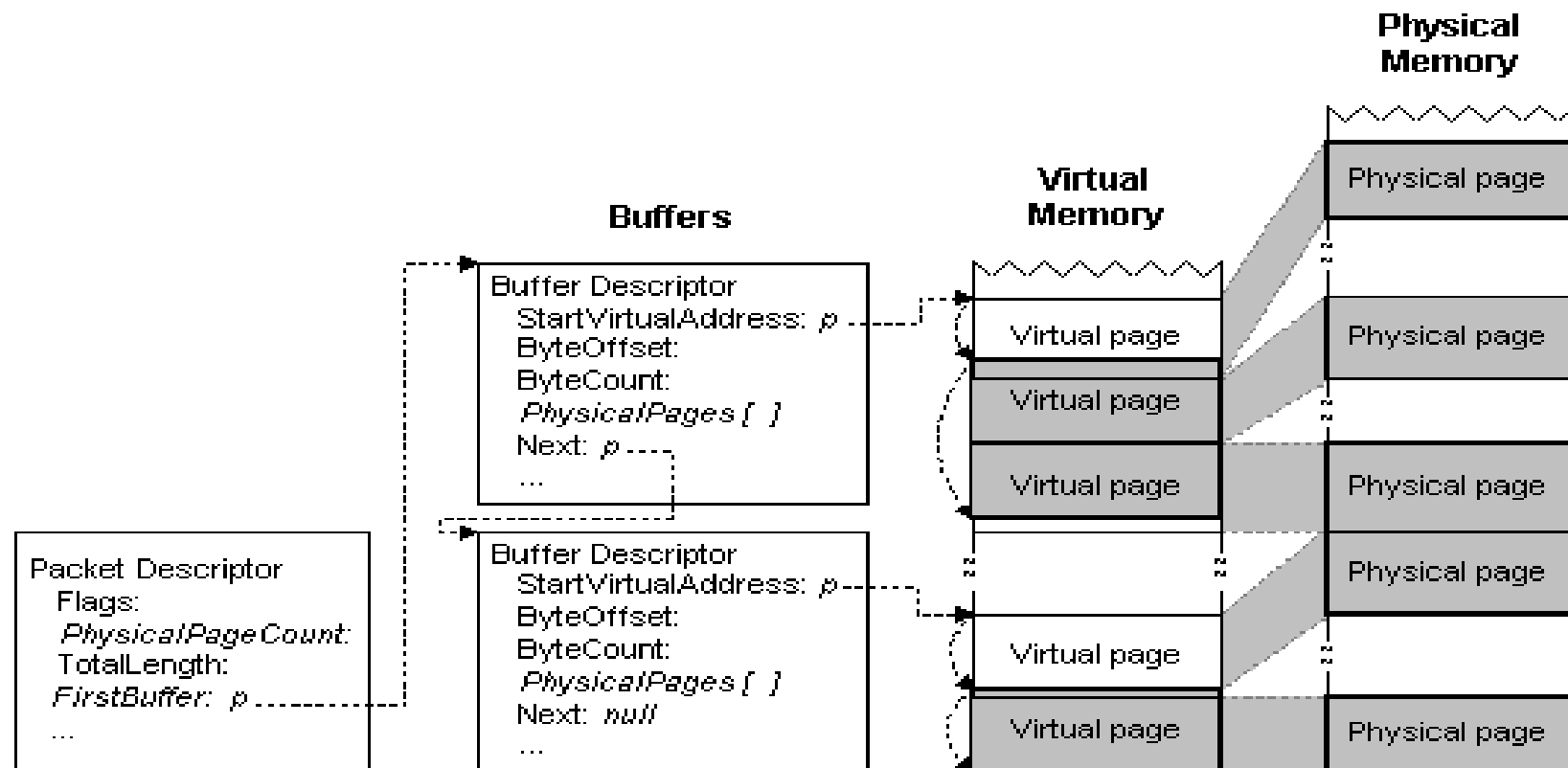


(Reference: <http://www.filseclab.com/eng/products/sourcetek.htm>)

NDIS Packet Structure

- A protocol driver allocates an NDIS packet (represented by NDIS_PACKET structure), fills it with data, and passes it to the next lower NDIS driver so that the data can be sent on the network.
- Some lowest level NIC drivers allocate packets to hold received data and pass the packet to interested higher-layer driver.

NDIS Packet Structure

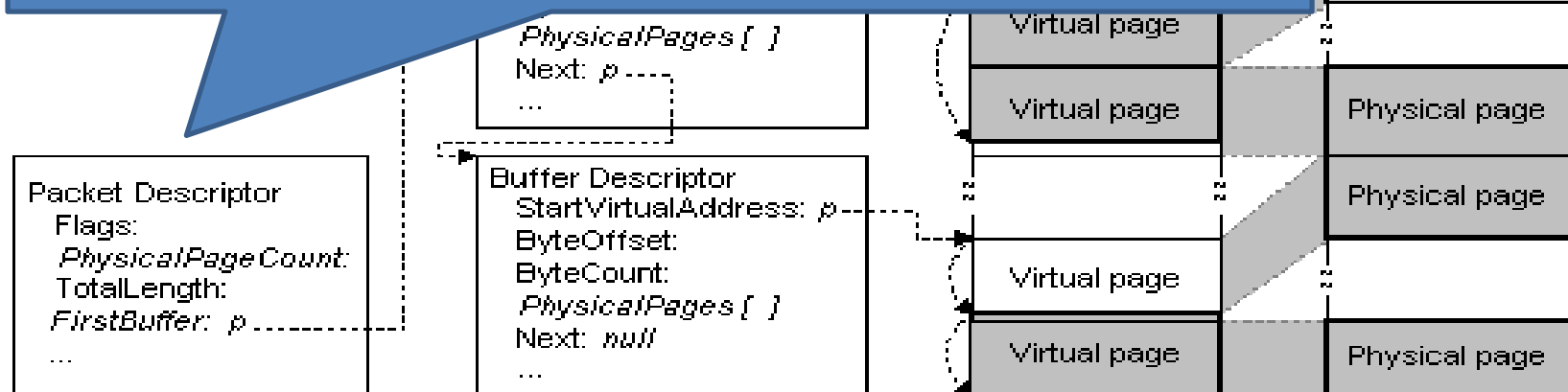


NDIS provides functions for allocating and manipulating the substructures that make up a packet

Reference: MSDN Microsoft

NDIS Packet Structure

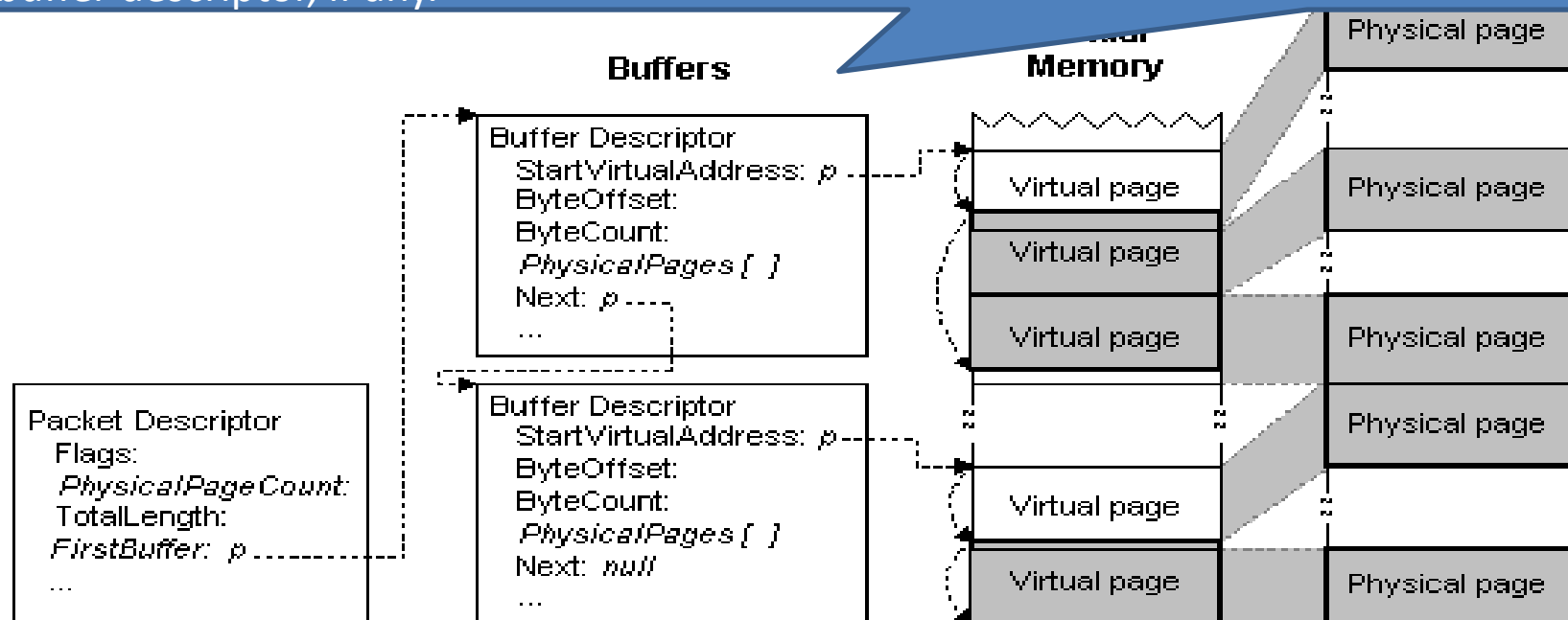
A packet descriptor - a set of flags associated with the packet and whose meaning is defined by a cooperating miniport driver(s) and protocol driver(s), the number of physical pages that contain the packet, the total length of the packet, and a pointer to the first buffer descriptor that maps the first buffer in the packet.



NDIS provides functions for allocating and manipulating the substructures that make up a packet

NDIS Packet Structure

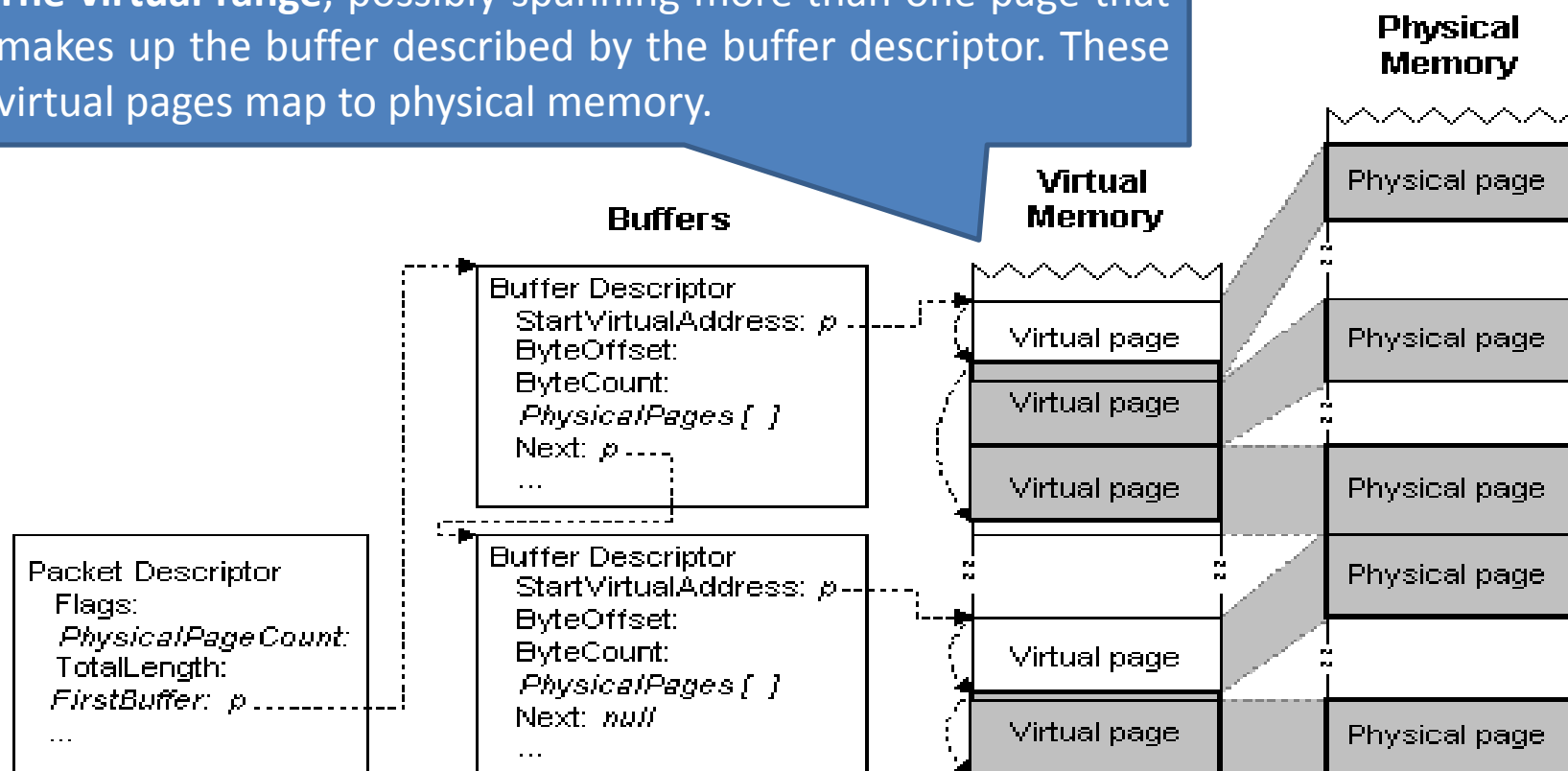
A set of buffer descriptors - A buffer descriptor describes the **starting virtual address** of each buffer, the buffer's byte offset into the page pointed to by the virtual address, the total number of bytes in the buffer and a pointer to the next buffer descriptor, if any.



NDIS provides functions for allocating and manipulating the substructures that make up a packet

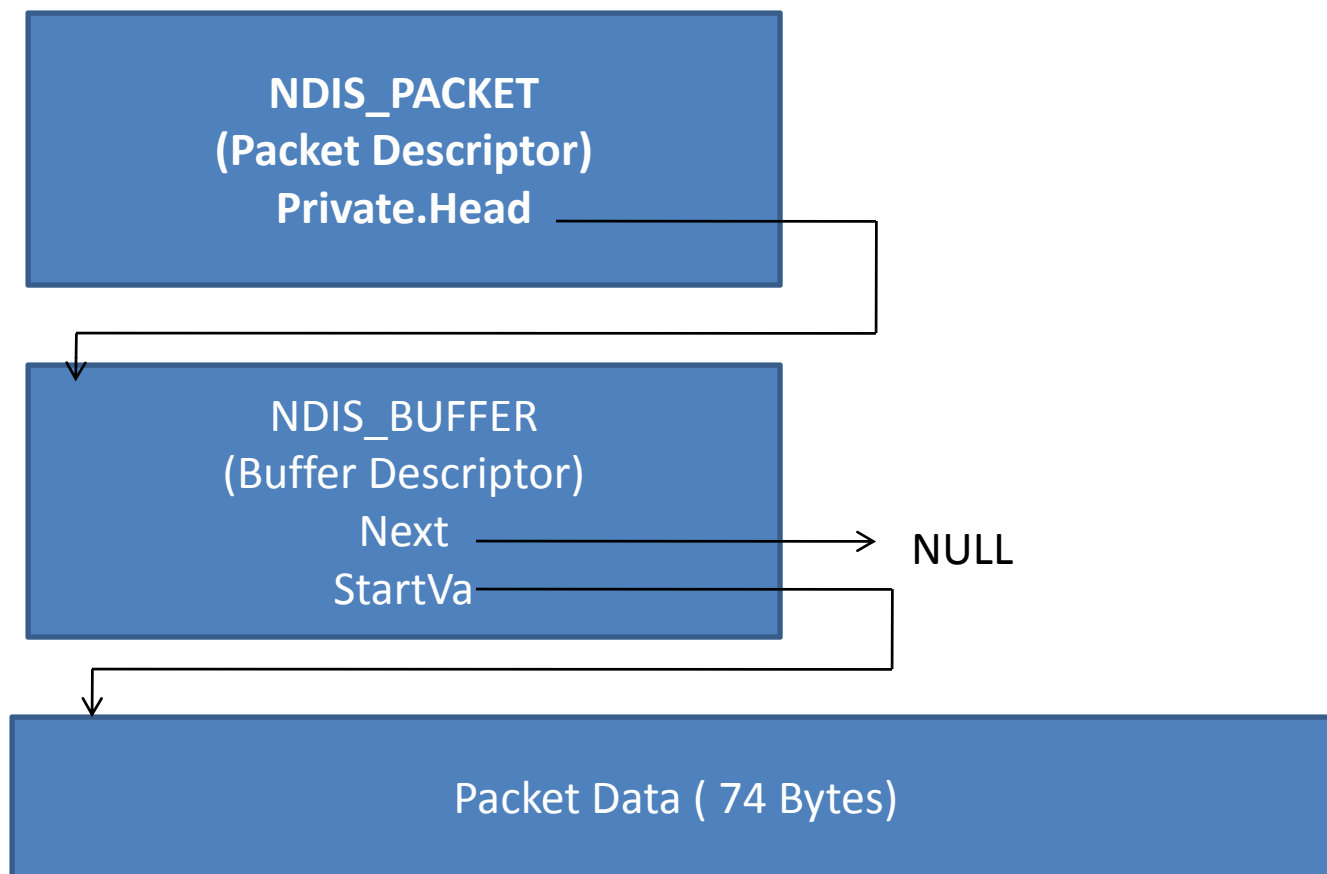
NDIS Packet Structure

The **virtual range**, possibly spanning more than one page that makes up the buffer described by the buffer descriptor. These virtual pages map to physical memory.



NDIS provides functions for allocating and manipulating the substructures that make up a packet

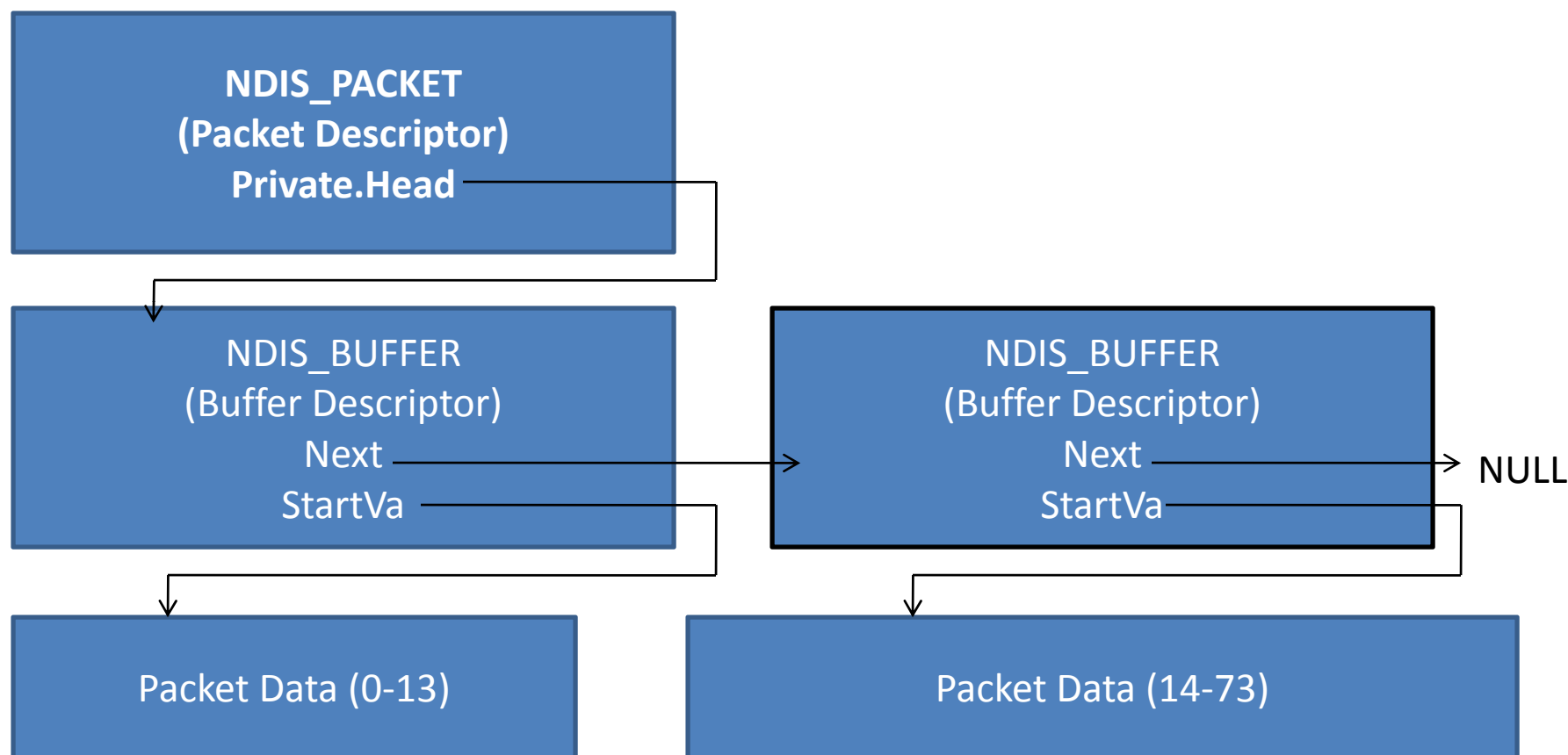
NDIS Packet Structure



Simple NDIS_PACKET Illustration

(Reference: PCAUSA)

NDIS Packet Structure



Multi-Buffer NDIS_PACKET Illustration

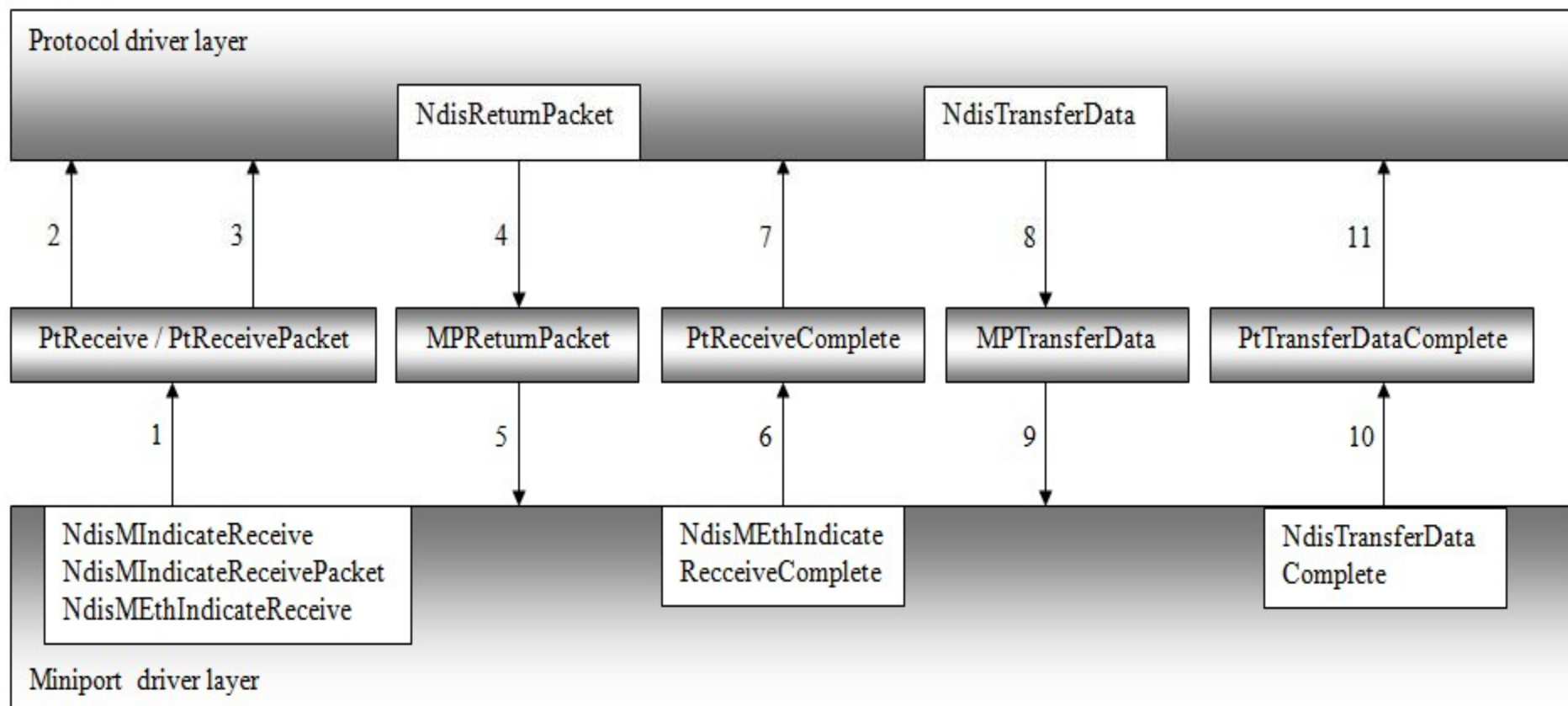
(Reference: PCAUSA)

Interpreting A NDIS Packet

- NDIS functions that need to inspect a NDIS_PACKET and the NDIS_BUFFER:
 - a. NdisQueryPacket: Returns information about a given packet descriptor
 - b. NdisQueryBufferSafe: Returns information about a given buffer descriptor
 - c. NdisGetNextBuffer: Returns the next buffer descriptor in a chain, given a pointer to the current buffer descriptor

Flowchart

Flowchart

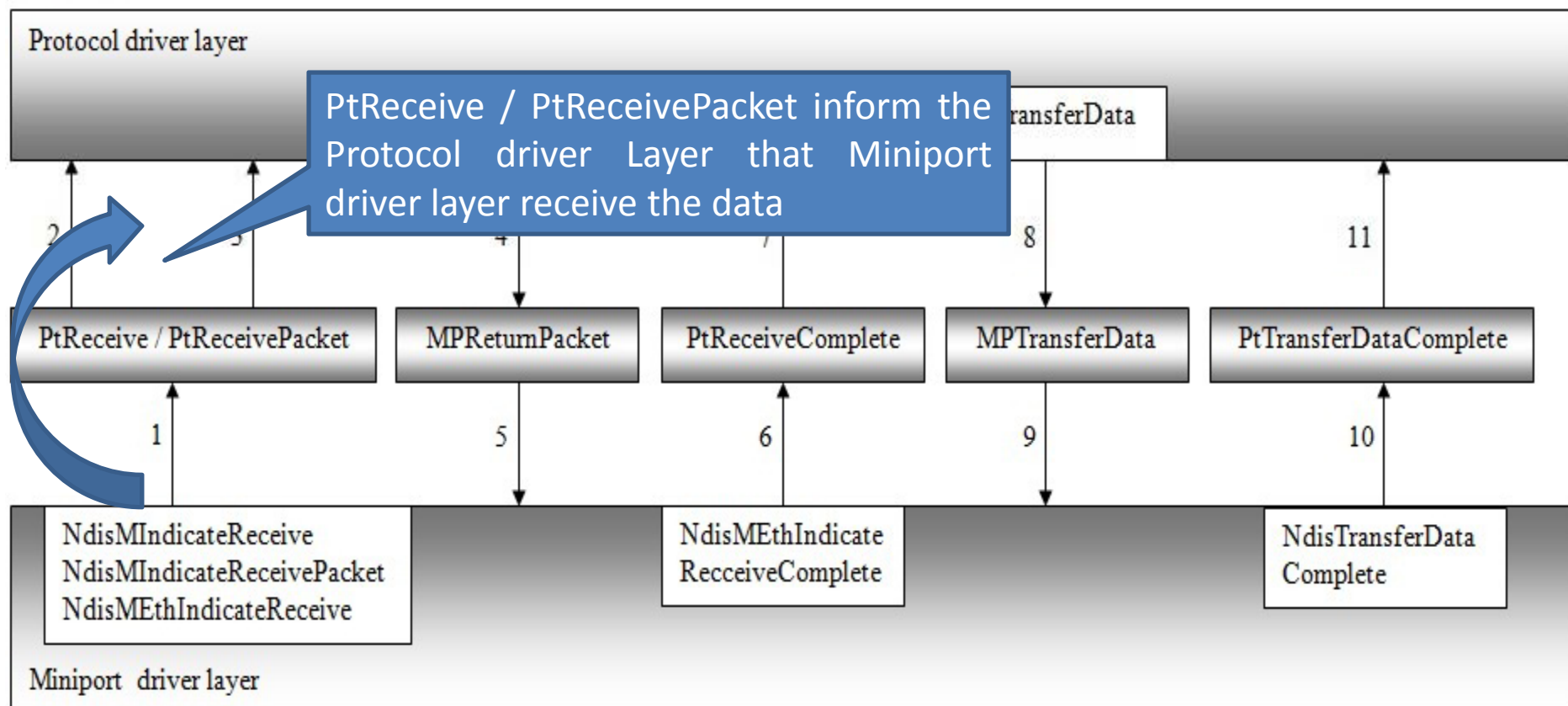


Flowchart of NDIS IM Driver Receive Network Packets

Reference: bbs.driverdevelop.com

Flowchart

Flowchart

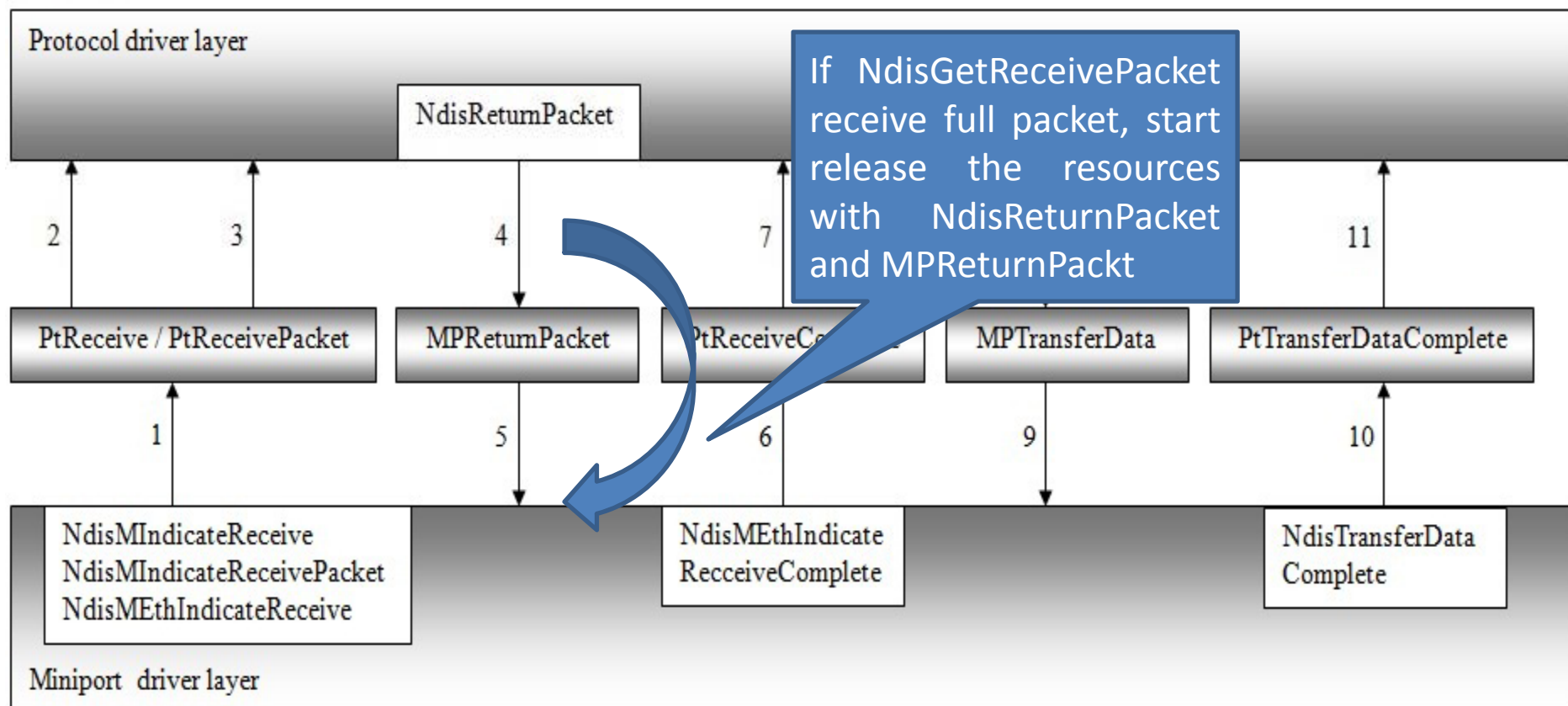


Flowchart of NDIS IM Driver Receive Network Packets

Reference: bbs.driverdevelop.com

Flowchart

Flowchart

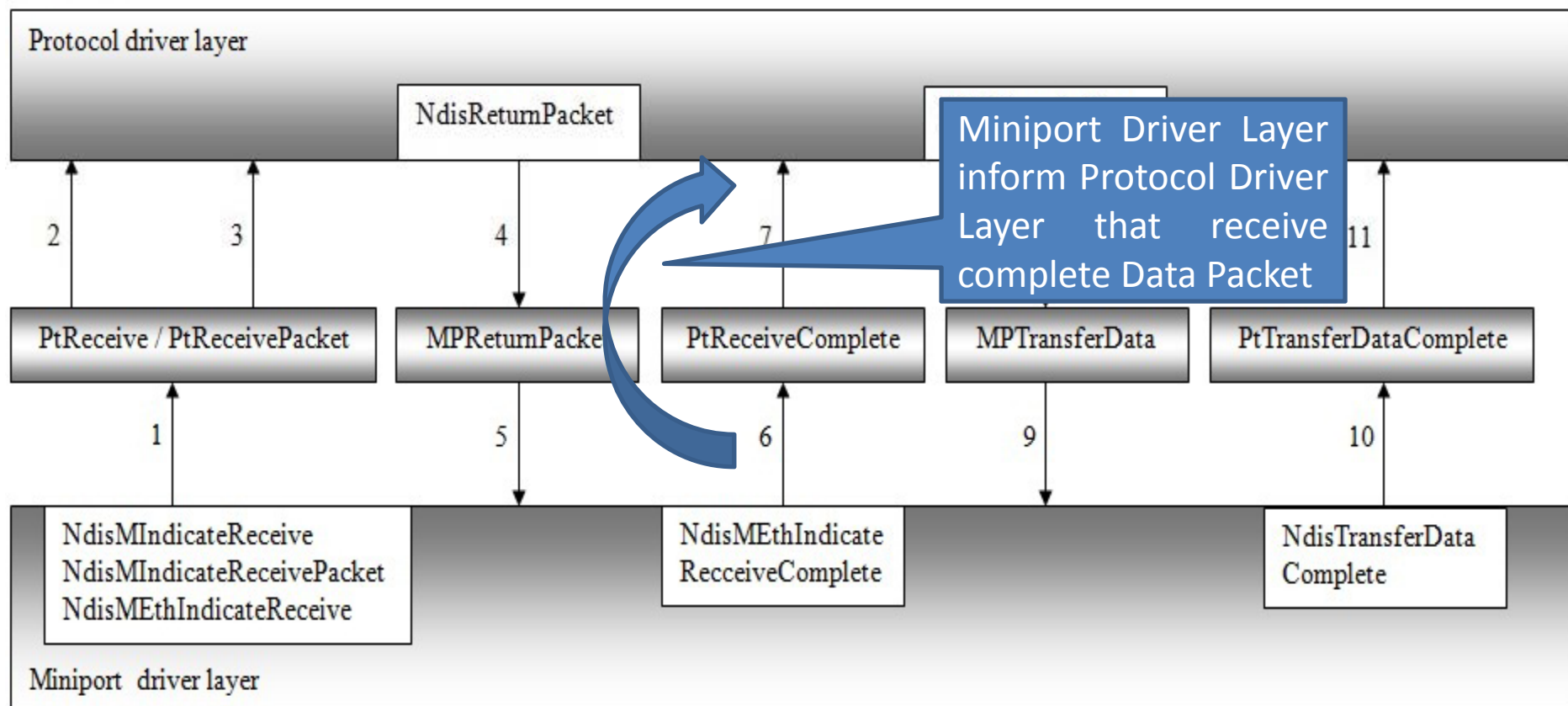


Flowchart of NDIS IM Driver Receive Network Packets

Reference: bbs.driverdevelop.com

Flowchart

Flowchart

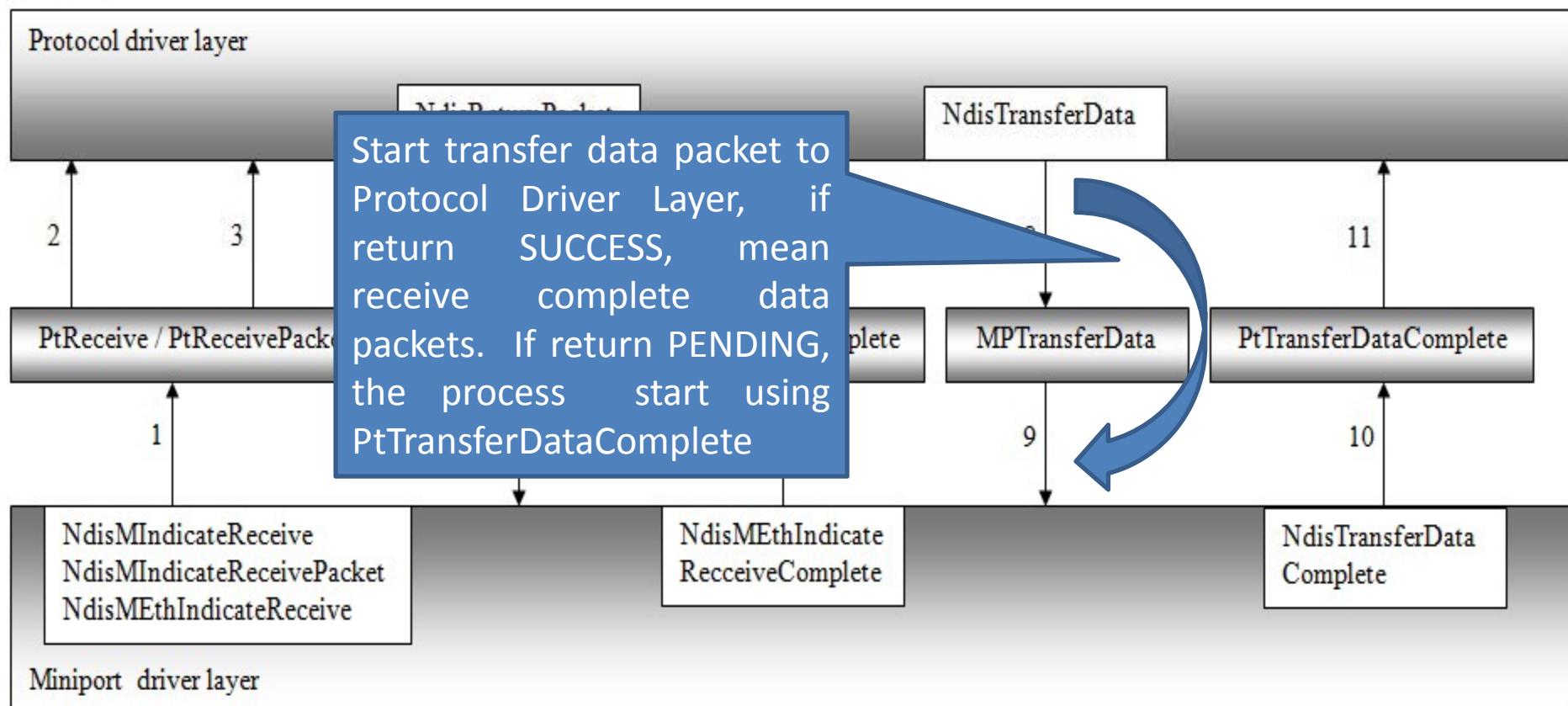


Flowchart of NDIS IM Driver Receive Network Packets

Reference: bbs.driverdevelop.com

Flowchart

Flowchart

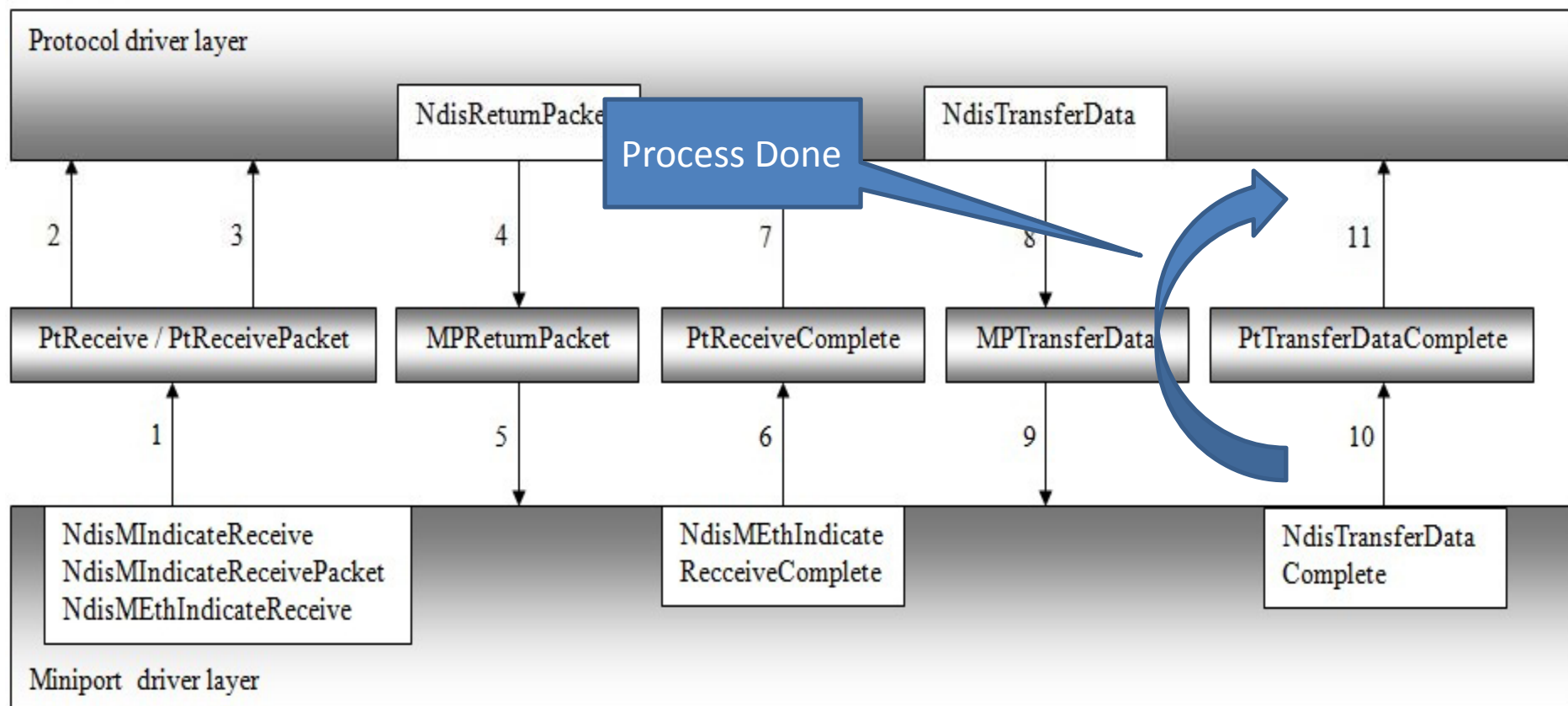


Flowchart of NDIS IM Driver Receive Network Packets

Reference: bbs.driverdevelop.com

Flowchart

Flowchart

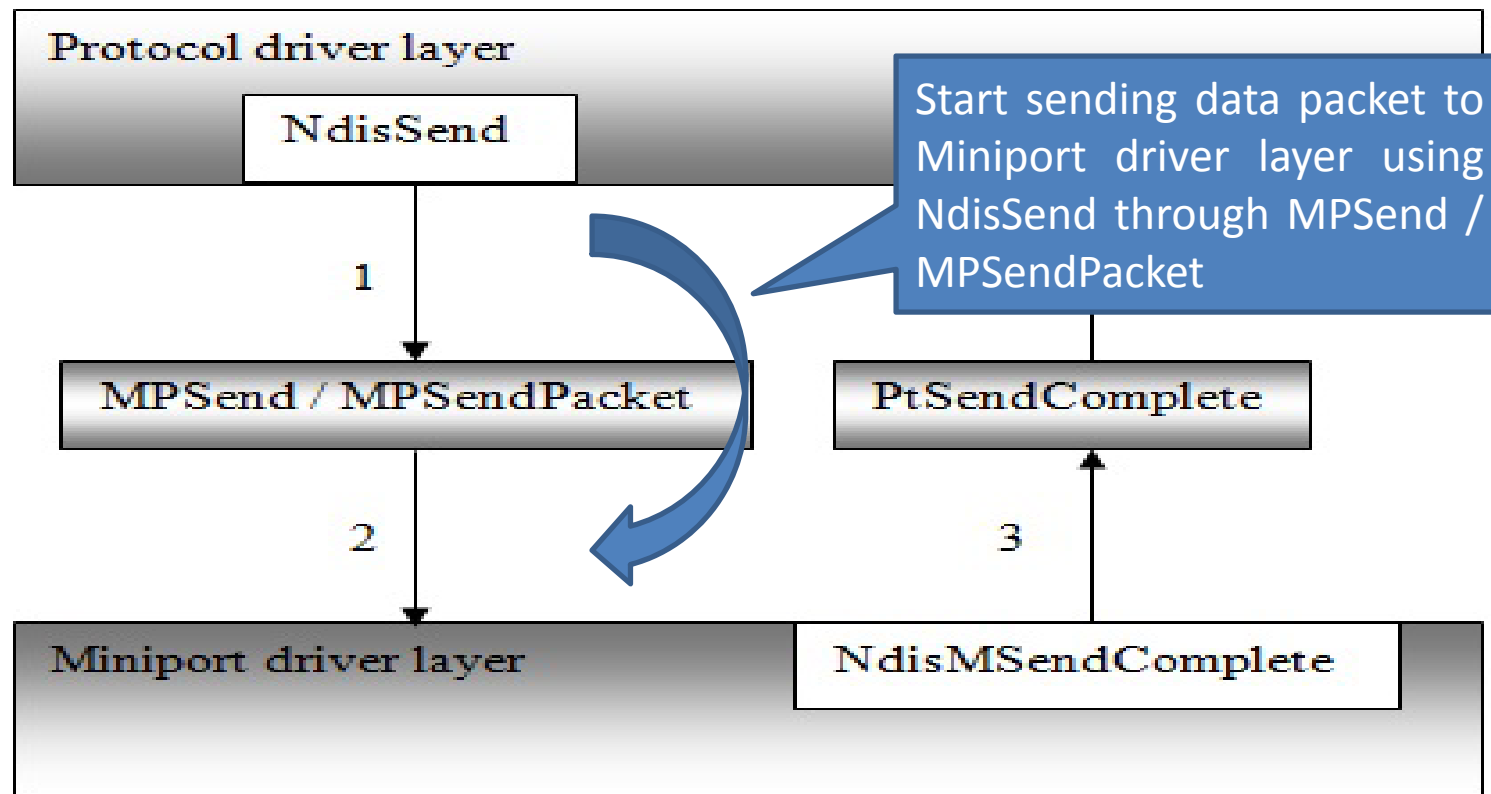


Flowchart of NDIS IM Driver Receive Network Packets

Reference: bbs.driverdevelop.com

Flowchart

Flowchart

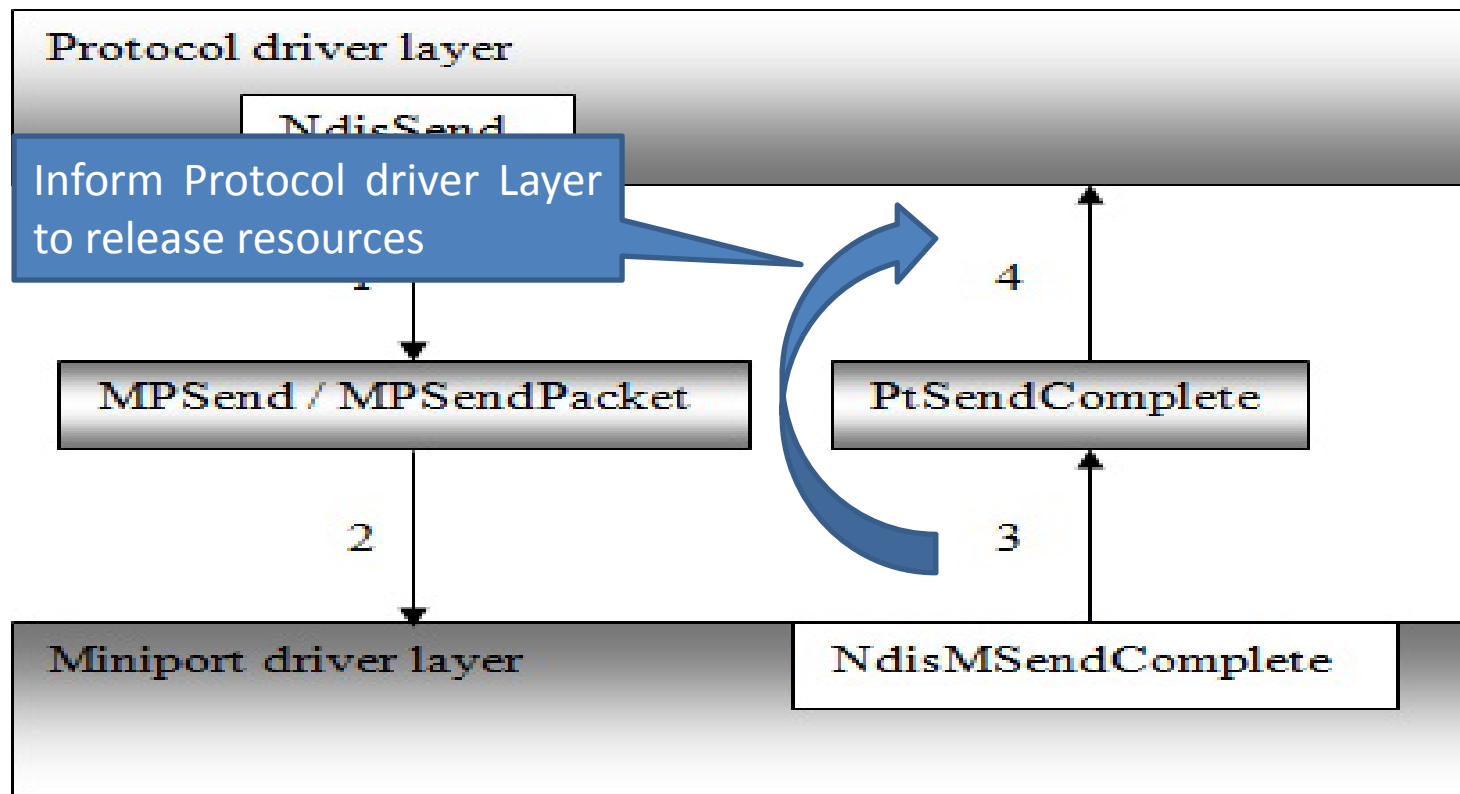


Flowchart of NDIS IM Driver Send Network Packets

Reference: bbs.driverdevelop.com

Flowchart

Flowchart

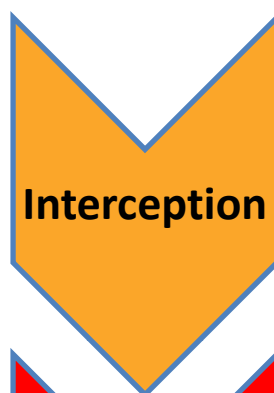


Flowchart of NDIS IM Driver Send Network Packets

Reference: bbs.driverdevelop.com

Interception & Blocking

Interception & Blocking



- Interception of sending and receiving network traffic packets



- Implementation of scanning and checking algorithm



- Passing or blocking network traffic packets

Interception & Blocking

Interception

- Interception of sending and receiving network traffic packets

Scanning & Checking

Actions

```

FILTER_STATUS AnalysisPacket(PNDIS_PACKET Packet, BOOLEAN bRecOrSend)
{
    FILTER_STATUS status = STATUS_PASS; //by default all passed
    PNDIS_BUFFER NdisBuffer, NdisNextBuffer;
    UINT TotalPacketLength = 0;
    UINT copysize = 0;
    UINT DataOffset = 0 ;
    UINT PhysicalBufferCount;
    UINT BufferCount;
    PCHAR pPacketContent = NULL;
    char* tcsPrintBuf = NULL;
    PCHAR tembuffer = NULL ;
    UINT j=0;
    UINT i;
    UINT k;
    PCHAR T="$1/1@1K1Q1t1z1";
    UINT totalLength;
    
```

Interception & Blocking

Interception

Scanning & Checking

Actions

394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444

```

TT[0] = -1;
TT[1]=0;
l=2;
k=0;

while(T[l] !='\0') {
    if (T[l-1]==T[k]) {
        TT[l] = k+1;
        k++;
        l++;
    }
    else if (k > 0) {
        k=TT[k];
    }
    else {
        TT[l] = 0;
        l++;
        k=0;
    }
}

if(pPacketContent[34]==0 && pPacketContent[35]==80)
{
    DbgPrint("Receive Http Port 80");
    for(i=54;i<=DataOffset;i++)
    {

        while(pPacketContent[i+j]!='\0' && T[j]!='\0')
        {
            if (pPacketContent[i+j] == T[j])
            {
                DbgPrint("Signature Match Success");
                ++j;
                if(T[j]=='\0')
                {
                    DbgPrint("myndis blocking - malicious traffic detected");
                    return STATUS_DROP;
                }
            }
            else
            {
                i += j -TT[j];
                if (j >0)
                    j += TT[j];
            }
        }
    }
}
    
```

Interception & Blocking

```
typedef struct _IPPacket{

    //Ethernet Header
    unsigned char  tar_hw_addr [6]; //6 bytes
    unsigned char  src_hw_addr [6]; //6 bytes
    unsigned char  H_frame_type;    //1 byte
    unsigned char  L_frame_type;    //1 byte

    //IP Header
    unsigned char  h_version;        //1 byte
    unsigned char  tos;               //1 byte
    unsigned short total_len;         //2 bytes
    unsigned short ident;             //2 bytes
    unsigned short frag_and_flags;    //2 bytes
    unsigned char  ttl;               //1 byte
    unsigned char  proto;             //1 byte
    unsigned short checksum;          //2 bytes
    unsigned char  sourceIP[4];       //4 bytes
    unsigned char  destIP[4];         //4 bytes

}
```

kd> db 82086050

82086050	00 0c 29 a3 6e 45 00 0c-29 f6 9c ae 08 00 45 00	..).nE..).E.
82086060	00 30 01 30 40 00 80 06-76 44 c0 a8 01 01 c0 a8	.0.0@...vD.....
82086070	01 02 00 50 04 0b 28 b5-e1 6d 66 be 8d 65 70 12	...P..(..mf...ep.
82086080	fa f0 02 29 00 00 02 04-05 b4 01 01 04 02 00 00	...).
82086090	09 00 10 1a 20 d0 2e 82-00 00 1d 00 00 00 00 00	
820860a0	00 00 00 00 00 00 38 00-00 00 00 00 00 00 00 00	8.....
820860b0	00 00 38 00 01 00 00 00-00 00 00 00 00 00 1d 00	..8.....
820860c0	01 00 00 00 00 00 00 00-80 00 00 00 00 00 00 00	

Interception & Blocking

```
{H_frame_type,L_frame_type}
    {0x08, 0x00 ->IP}, {0x08, 0x06 -> ARP}, {0x08, 0x35 -> RARP}

{proto}
    {0x1->ICMP}, {0x2->IGMP}, {0x6->TCP}, {0x11->UDP}
```

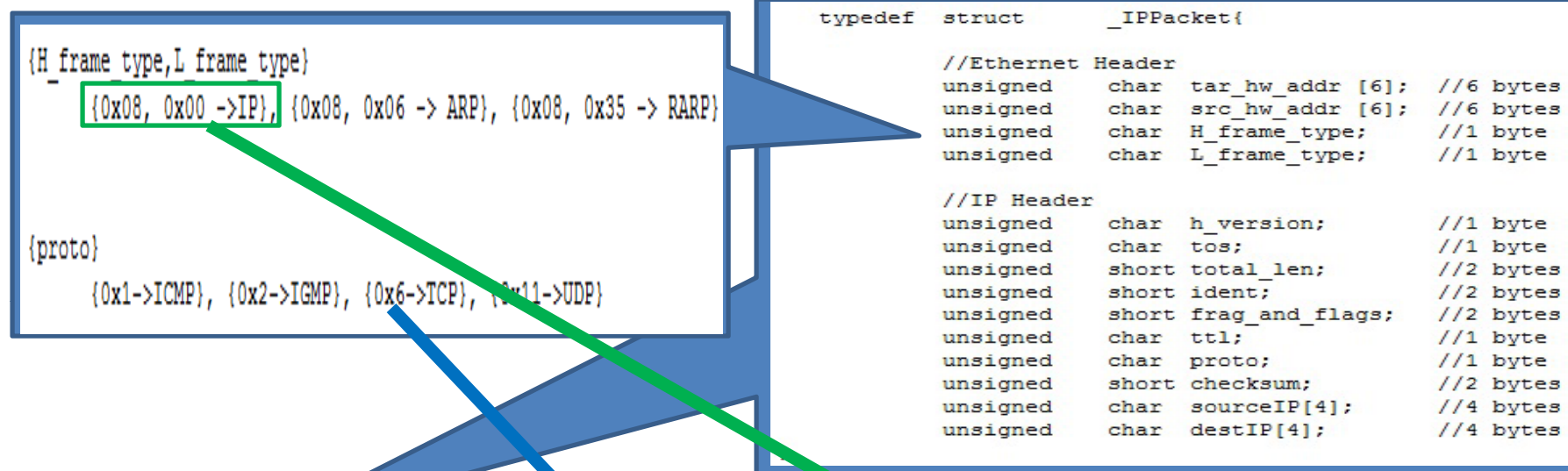
```
typedef struct _IPPacket{

    //Ethernet Header
    unsigned char  tar_hw_addr [6]; //6 bytes
    unsigned char  src_hw_addr [6]; //6 bytes
    unsigned char  H_frame_type;    //1 byte
    unsigned char  L_frame_type;    //1 byte

    //IP Header
    unsigned char  h_version;       //1 byte
    unsigned char  tos;              //1 byte
    unsigned short total_len;        //2 bytes
    unsigned short ident;           //2 bytes
    unsigned short frag_and_flags;  //2 bytes
    unsigned char  ttl;             //1 byte
    unsigned char  proto;           //1 byte
    unsigned short checksum;        //2 bytes
    unsigned char  sourceIP[4];     //4 bytes
    unsigned char  destIP[4];       //4 bytes
}
```

```
kd> db 82086050
82086050 00 0c 29 a3 6e 45 00 0c-29 f6 9c ae 08 00 45 00 ..).nE..).E.
82086060 00 30 01 30 40 00 80 06-76 44 c0 a8 01 01 c0 a8 .0.0@...vD....
82086070 01 02 00 50 04 0b 28 b5-e1 6d 66 be 8d 65 70 12 ...P..(..mf...ep.
82086080 fa f0 02 29 00 00 02 04-05 b4 01 01 04 02 00 00 ...).
82086090 09 00 10 1a 20 d0 2e 82-00 00 1d 00 00 00 00 00 .....
820860a0 00 00 00 00 00 00 38 00-00 00 00 00 00 00 00 .....8.....
820860b0 00 00 38 00 01 00 00 00-00 00 00 00 00 1d 00 ..8.....
820860c0 01 00 00 00 00 00 00 00-80 00 00 00 00 00 00 .....
```

Interception & Blocking



```
kd> db 82086050
82086050 00 0c 29 a3 6e 45 00 0c-29 f6 9c ae 08 00 45 00 ..).nE..).E.
82086060 00 30 01 30 40 00 80 06-76 44 c0 a8 01 01 c0 a8 .0.0@...vD....
82086070 01 02 00 50 04 0b 28 b5-e1 6d 66 be 8d 65 70 12 ...P..(..mf...ep.
82086080 fa f0 02 29 00 00 02 04-05 b4 01 01 04 02 00 00 ...).
82086090 09 00 10 1a 20 d0 2e 82-00 00 1d 00 00 00 00 00 .....
820860a0 00 00 00 00 00 00 38 00-00 00 00 00 00 00 00 .....8.....
820860b0 00 00 38 00 01 00 00 00-00 00 00 00 00 00 1d 00 ..8.....
820860c0 01 00 00 00 00 00 00 00-80 00 00 00 00 00 00 .....
```

Interception & Blocking

```
typedef struct _tcphdr
{
    USHORT th_sport; //source port (2 byte)
    USHORT th_dport; //destination port (2 byte)
    unsigned int th_seq; //sequence number (4 byte)
    unsigned int th_ack; //acknowledgment number (4 byte)
    unsigned char th_lenres; //Data Offset+Reserved
    unsigned char th_flag; // (th_lenres+th_flag = 2 byte)
    USHORT th_win; //Window (2 byte)
    USHORT th_sum; //Checksum (2 byte)
    USHORT th_urp; //Urgent Pointer (2 byte)
}TCP_HEADER;
```

```
kd> db 82086050
82086050 00 0c 29 a3 45 00 0c-29 f6 9c ae 08 00 45 00 ..).nE..).E.
82086060 00 30 01 20 40 00 80 06-76 44 c0 a8 01 01 c0 a8 .0.0@...vD...
82086070 01 02 00 50 04 0b 28 b5-e1 6d 66 be 8d 65 70 12 ...P..(.mf.ep.
82086080 fa f0 02 29 00 00 02 04-05 b4 01 01 04 02 00 00 ...).....
82086090 09 00 10 1a 20 d0 2e 82-00 00 1d 00 00 00 00 00 .....
820860a0 00 00 00 00 00 00 38 00-00 00 00 00 00 00 00 .....8.....
820860b0 00 00 38 00 01 00 00 00-00 00 00 00 00 1d 00 ..8.....
820860c0 01 00 00 00 00 00 00 00-80 00 00 00 00 00 00 .....
```

Interception & Blocking

```
typedef struct _tcphdr
{
    USHORT th_sport; //source port (2 byte)
    USHORT th_dport; //destination port (2 byte)
    unsigned int th_seq; //sequence number (4 byte)
    unsigned int th_ack; //acknowledgment number (4 byte)
    unsigned char th_lenres; //Data Offset+Reserved
    unsigned char th_flag; // (th_lenres+th_flag = 2 byte)
    USHORT th_win; //Window (2 byte)
    USHORT th_sum; //Checksum (2 byte)
    USHORT th_urp; //Urgent Pointer (2 byte)
}TCP_HEADER;
```

kd> db 82086050

82086050	00 0c 29 a3 45 00 0c-29 f6 9c ae 08 00 45 00	..).nE..).E.
82086060	00 30 01 20 40 00 80 06-76 44 c0 a8 01 01 c0 a8	.0.0@...vD.....
82086070	01 02 00 50 04 0b 28 b5-e1 6d 66 be 8d 65 70 12	...P..(..mf...ep.
82086080	fa f0 02 29 00 00 02 04-05 b4 01 01 04 02 00 00	...).E.....
82086090	09 00 10 1a 20 d0 2e 82-00 00 1d 00 00 00 00 00	8.....
820860a0	00 00 00 00 00 00 38 00-00 00 00 00 00 00 00 00	8.....
820860b0	00 00 38 00 01 00 00 00-00 00 00 00 00 00 1d 00	..8.....
820860c0	01 00 00 00 00 00 00 00-80 00 00 00 00 00 00 00	

Interception & Blocking

Interception

Scanning & Checking

Actions

```

} __finally
{
    if(pPacketContent)NdisFreeMemory(pPacketContent, 0, 0);
    if(tcsPrintBuf)NdisFreeMemory(tcsPrintBuf, 0, 0);
}
return STATUS_PASS;
}
    
```

- Passing or blocking network traffic packets

Installation of myndis driver

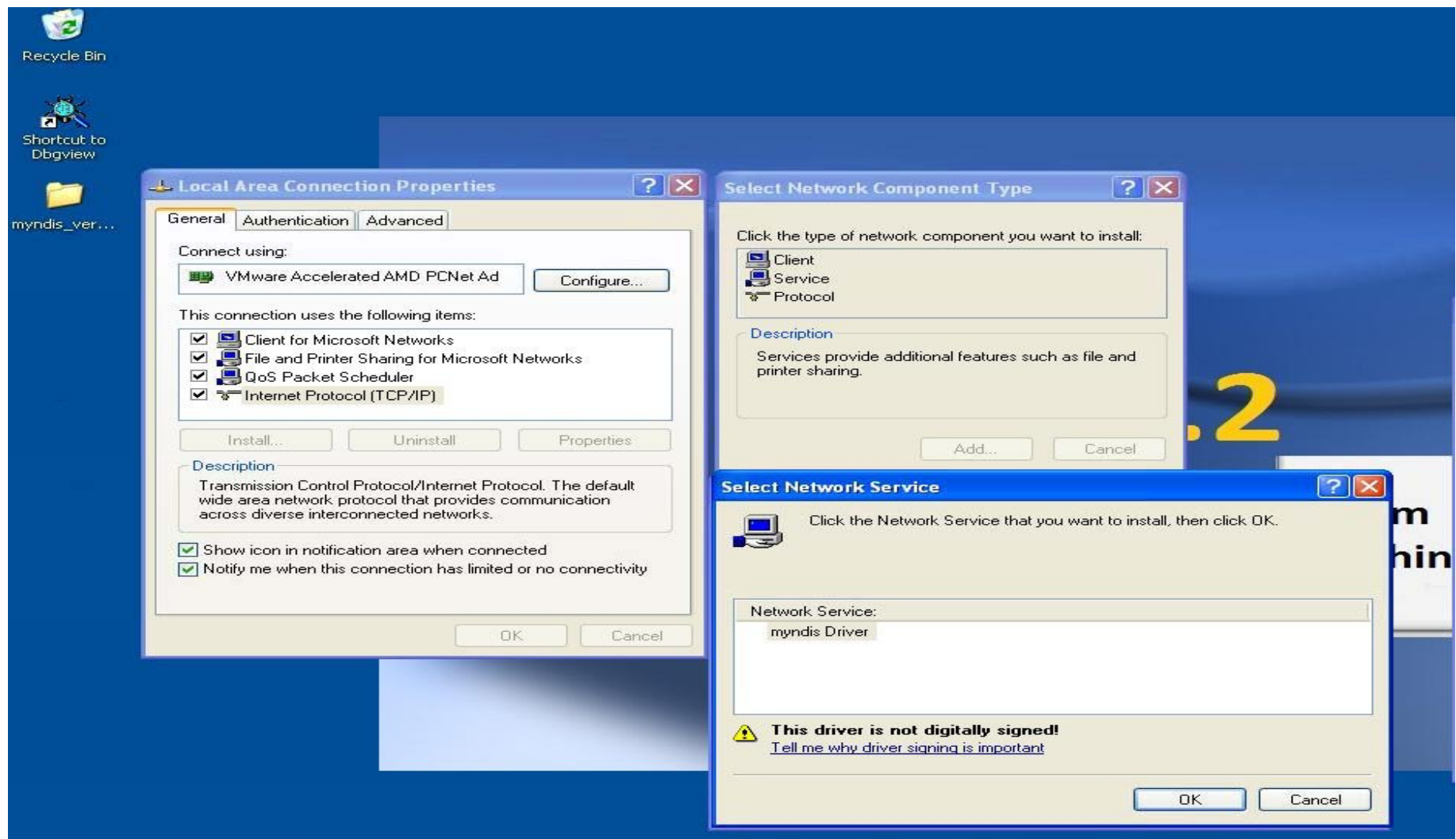
Installation

- Installation of myndis driver on Windows XP, 3 files are needed: netsf.inf, netsf_m.inf and myndis.sys.
- Open 'Local Area Connections' and choose 'Properties'.
- Right-click on the relevant Local Area Connection icon and choose 'Properties' again.

Installation

- Select 'Install' -> 'Service' -> 'Add' -> 'Have Disk', browse to the directory containing the 3 files listed in previous slides. Click 'OK'
- 'myndis Driver' should appear in a list of Network Services.
- Ignore the warning regarding installation of unsigned files

Installation



Installation



myndis Driver is installed and enabled

Interception and Blocking

DebugView on \\\CLEE-8C5BD4C23 (local)

#	Time	Debug Print
363	0.06538345	Signature Match Success
364	0.06555945	Receive Http Port 80
365	0.06558347	Signature Match Success
366	0.06559968	Signature Match Success
367	0.06561811	Signature Match Success
368	0.06570975	Send TCP packet
369	0.06592430	Receive Http Port 80
370	0.06594302	Signature Match Success
371	0.06596090	Signature Match Success
372	0.06611315	Receive Http Port 80
373	0.06613019	Signature Match Success
374	0.06620479	Signature Match Success
375	0.06622574	Signature Match Success
376	0.06624193	Signature Match Success
377	0.06633162	Send TCP packet
378	0.06650287	Receive Http Port 80
379	0.06652130	Signature Match Success
380	0.06653890	Signature Match Success
381	0.06660483	Receive Http Port 80
382	0.06667496	Signature Match Success
383	0.06669367	Signature Match Success
384	0.06707416	Send TCP packet
385	0.06724821	Receive Http Port 80
386	0.06726637	Signature Match Success
387	0.06728397	Signature Match Success
388	0.06730073	Signature Match Success
389	0.06731749	Signature Match Success
390	0.06733342	Signature Match Success
391	0.12058767	Send TCP packet
392	0.12113663	Send TCP packet
393	0.27255335	Send TCP packet
394	0.28155395	Receive Http Port 80
395	0.28161597	Receive Http Port 80
396	0.28165928	Send TCP packet
397	0.28481162	Send TCP packet
398	18.56476212	Send TCP packet
399	18.56510735	Receive Http Port 80
400	18.56515312	Send TCP packet
401	18.56546783	Send TCP packet
402	18.56712341	Receive Http Port 80
403	18.56713486	Signature Match Success
404	18.56718254	Receive Http Port 80
405	18.56722832	Send TCP packet
406	18.56745148	Receive Http Port 80
407	18.56748009	Receive Http Port 80
408	18.56751251	Send TCP packet
409	24.09462929	Receive Http Port 80
410	24.09480476	Send TCP packet

Index of / - Windows Internet Explorer
http://192.168.1.1/

0% of no_virus.exe from 192.168.1.1 Compl...

Getting File Information:
no_virus.exe from 192.168.1.1

Estimated time left
Download to:

File Download - Security Warning

Do you want to run or save this file?

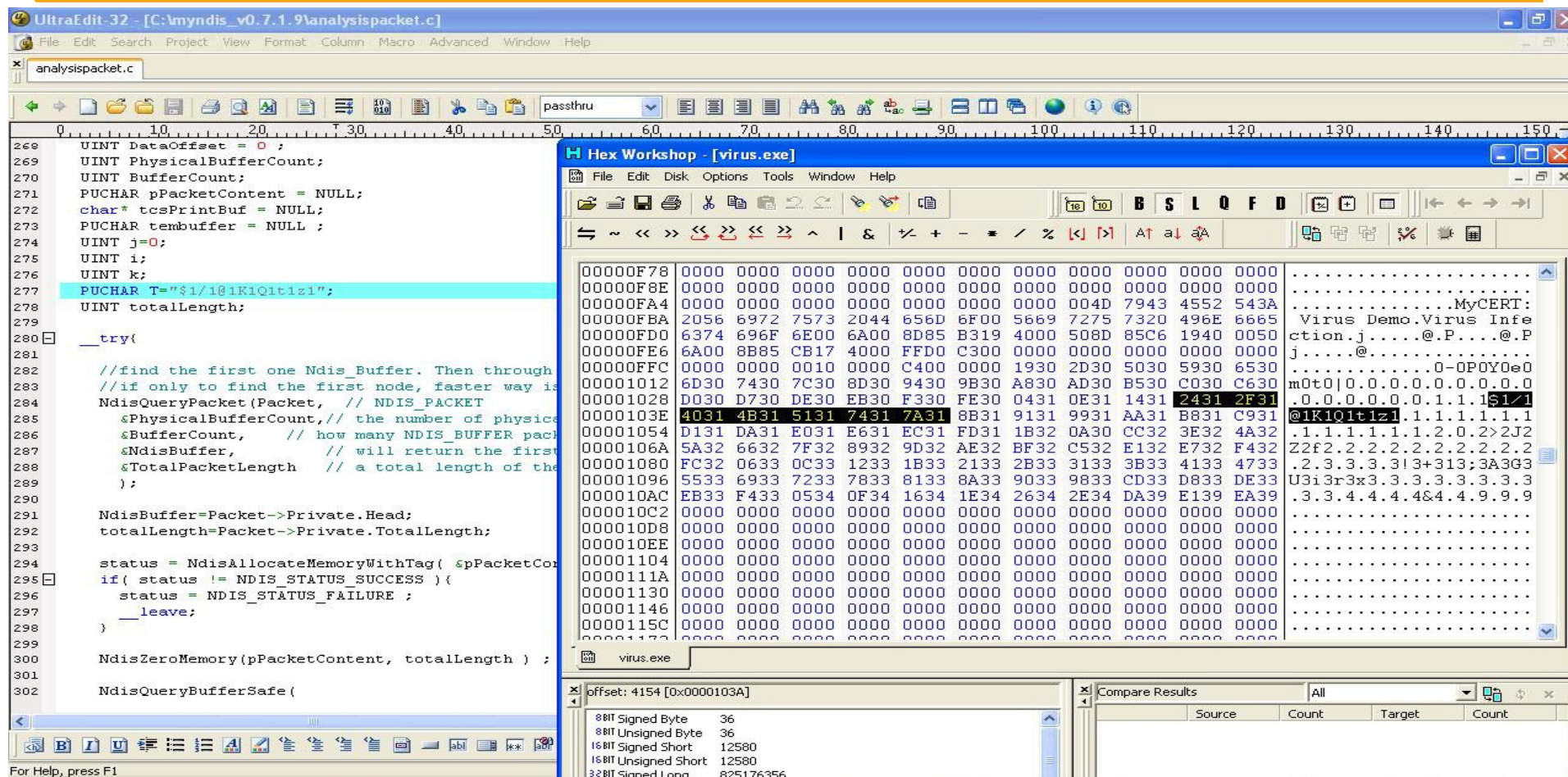
Name: no_virus.exe
Type: Application, 4.00KB
From: 192.168.1.1

Run Save Cancel

While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not run or save this software. [What's the risk?](#)

Executable file (no_virus.exe) is scanned by myndis driver

Interception & Blocking



Signature of malicious executable file was created for KMP
pattern matching action

Interception & Blocking

The screenshot displays two windows. The top window is 'DebugView on \VLCLÉE-8C5BD4C23 (local)', showing a list of network events. A red box highlights the following entries:

#	Time	Debug Print
578	72.12615967	Receive Http Port 80
579	72.12617493	Signature Match Success
580	72.12618256	Signature Match Success
581	72.12619781	Signature Match Success
582	72.12620544	Signature Match Success
583	72.12622070	Signature Match Success
584	72.12622833	Signature Match Success
585	72.12623596	Signature Match Success
586	72.12625122	Signature Match Success
587	72.12625885	Signature Match Success
588	72.12627411	Signature Match Success
589	72.12628174	Signature Match Success
590	72.12628937	Signature Match Success
591	72.12629700	Signature Match Success
592	72.12631226	Signature Match Success
593	72.12631989	myndis blocking - malicious traffic detected

The bottom window is 'Index of / - Windows Internet Explorer', showing a directory listing for 'http://192.168.1.1/'. The listing includes:

Name	Last modified	Size	Description
exploit.swf	12-May-2010 22:40	1.5K	
mycert.swf	03-Feb-2009 22:30	109K	
no_virus.exe	11-Aug-2009 17:21	4.0K	Win32 Executable
virus.exe	11-Aug-2009 17:36	8.0K	Win32 Executable

A 'File Download' dialog box is open, showing 'Getting File Information: virus.exe from 192.168.1.1'. The dialog includes fields for 'Estimated time left', 'Download to:', and 'Transfer rate:', along with a checkbox 'Close this dialog box when download completes' and buttons 'Open', 'Open Folder', and 'Cancel'.

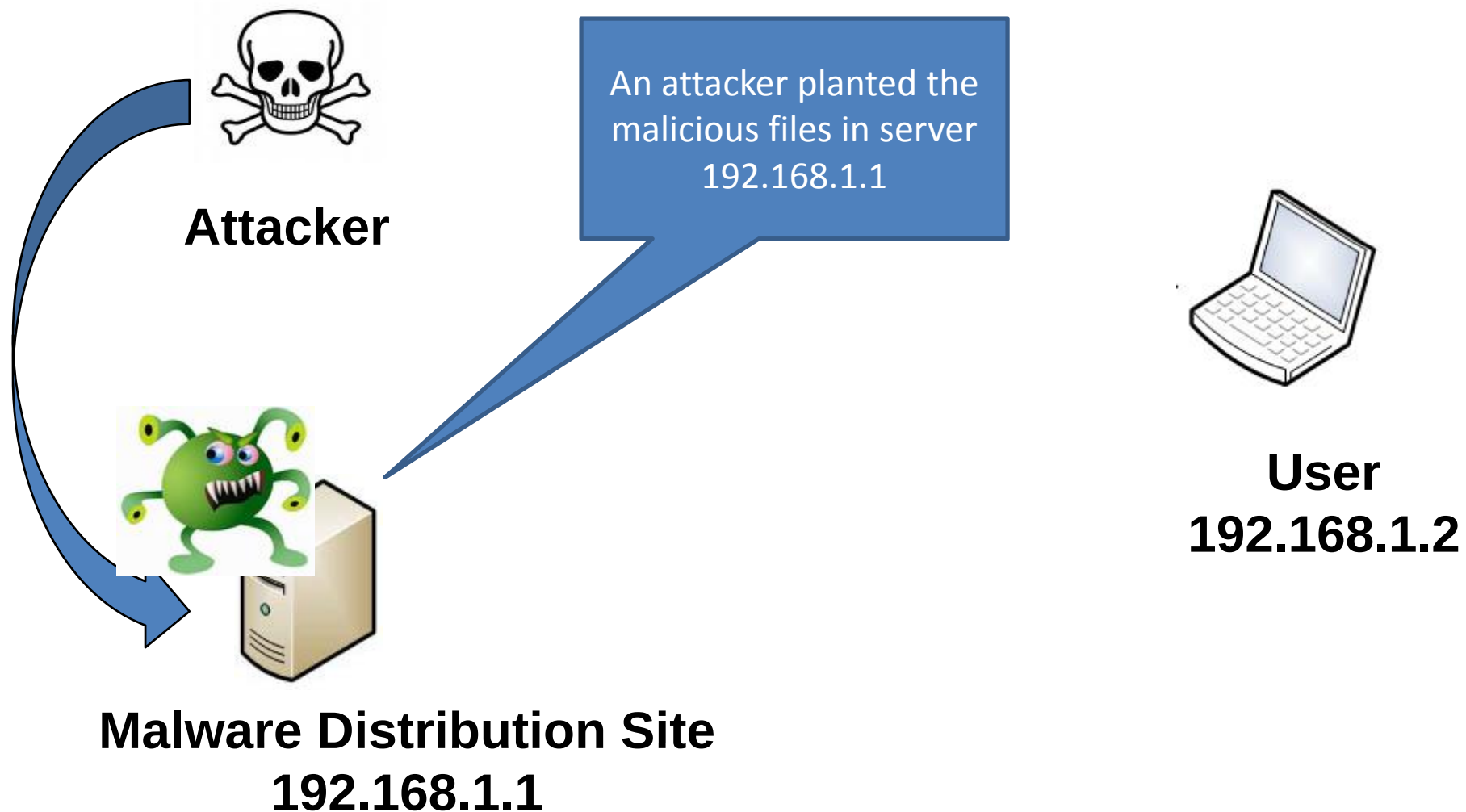
Malicious executable file (virus.exe) successful blocked by myndis
Driver

DEMO – Blocking Exploit Payload on the fly

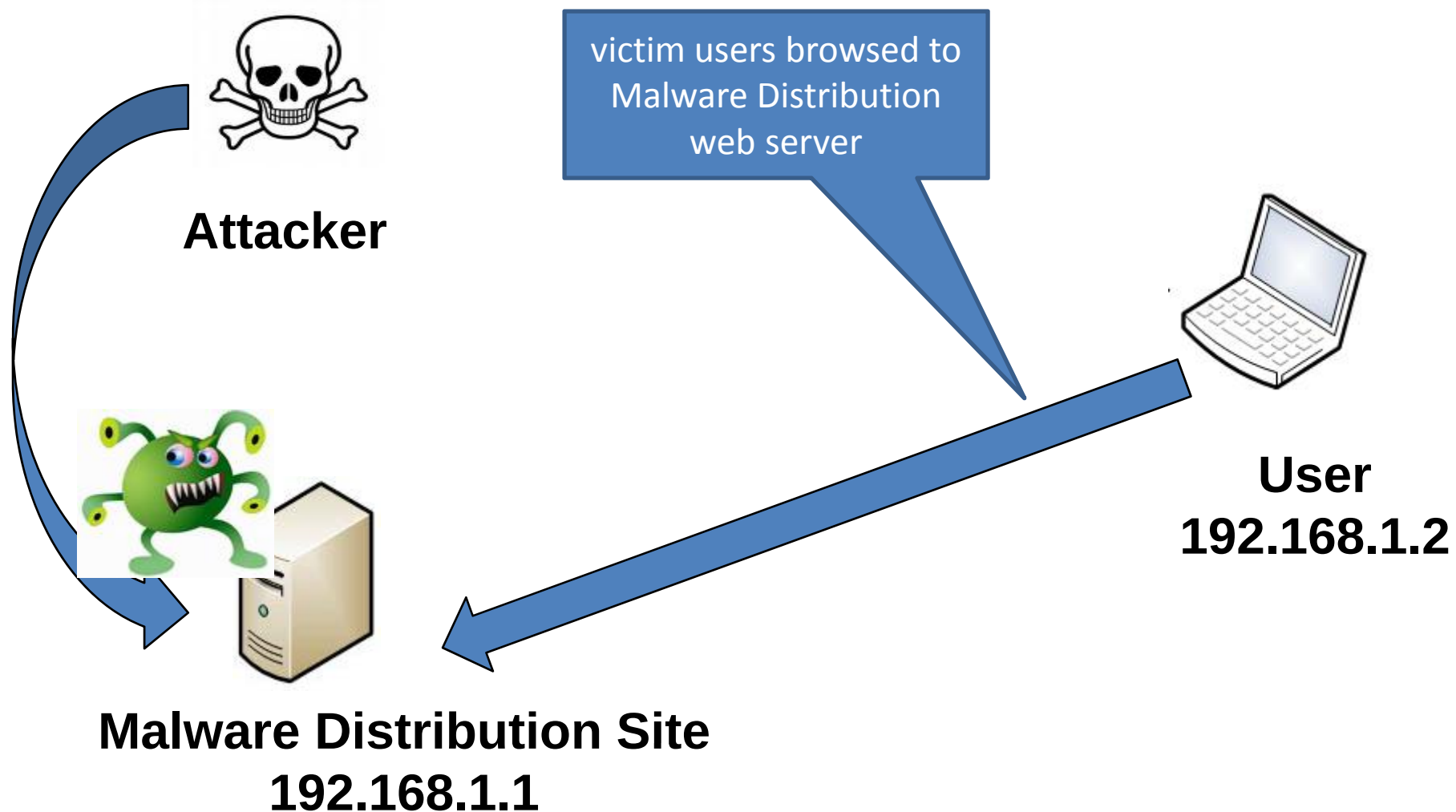
Scenario of the Demonstration

- An example of Integer Overflow in Adobe Flash Player 9.0.115.0 and earlier – CVE-27007-0071 (“old” vulnerability)
- Malware sample as a payload for Flash exploitation was coded (virus.exe)
- How the NDIS Intermediate Driver success to intercept and block the Adobe Flash vulnerability exploitation process

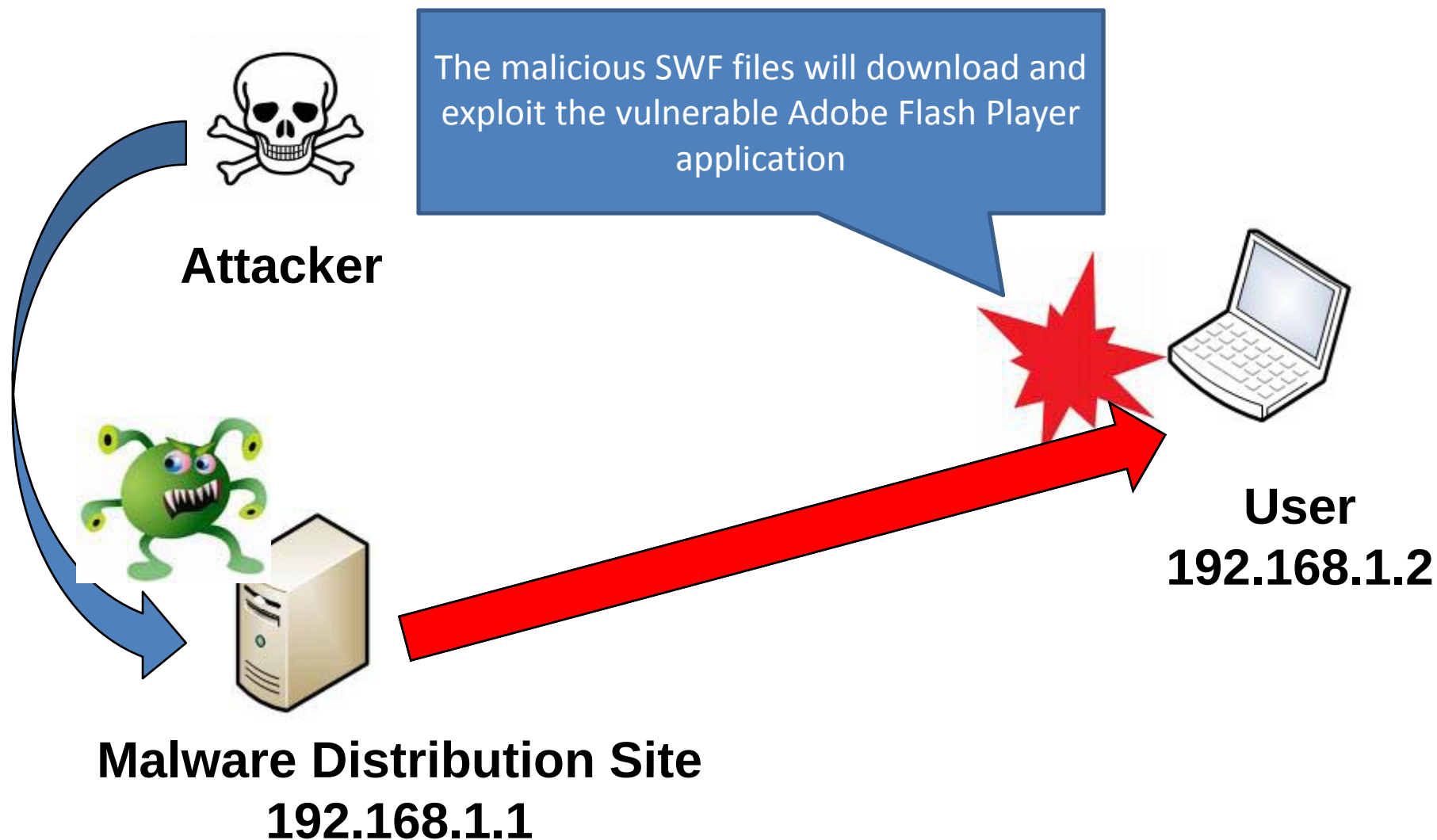
Scenario of the Demonstration



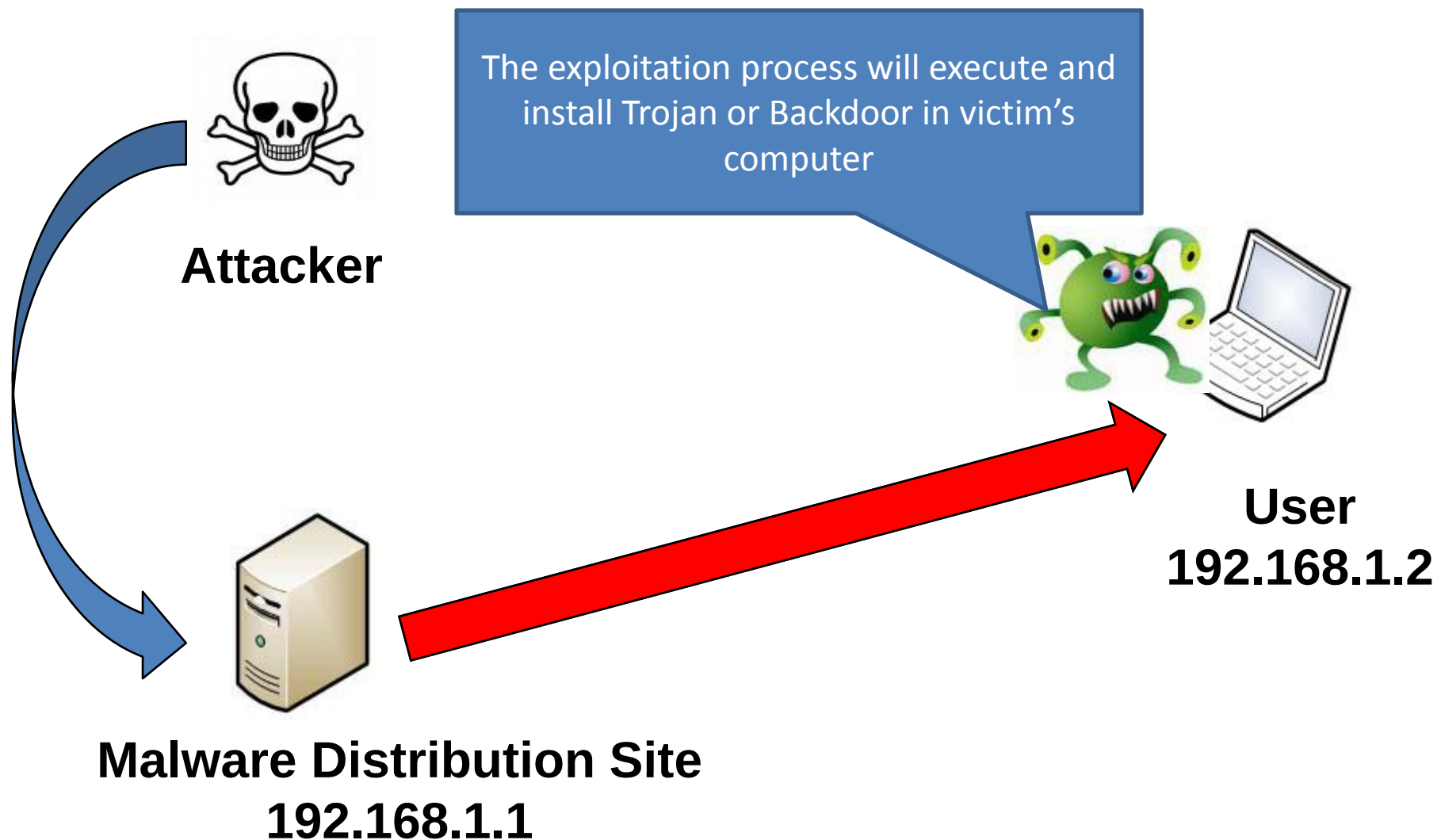
Scenario of the Demonstration



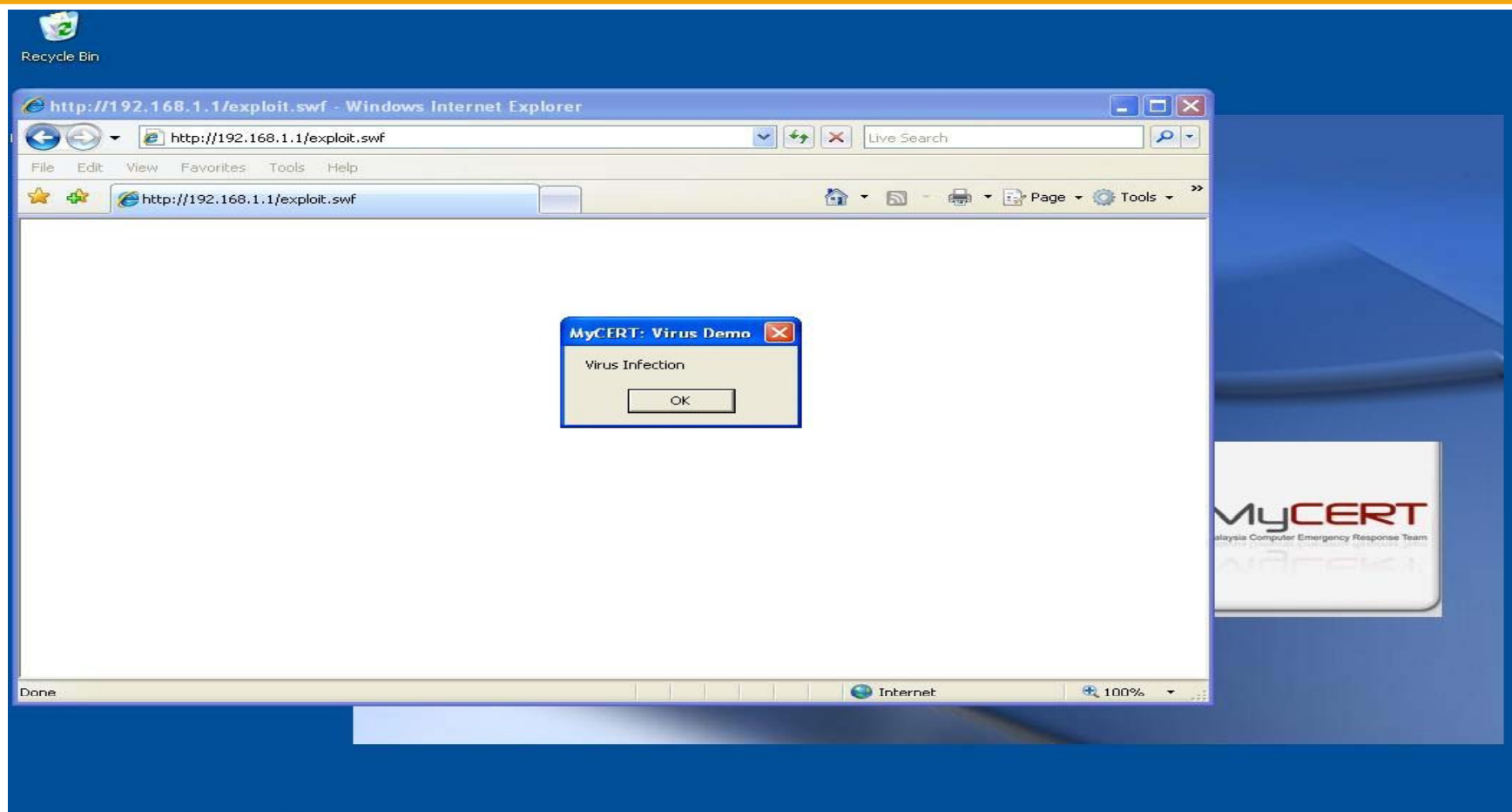
Scenario of the Demonstration



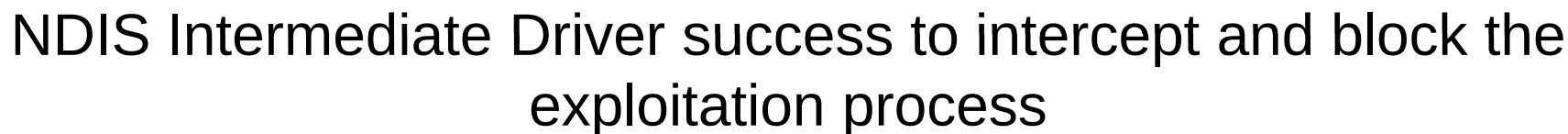
Scenario of the Demonstration



Interception & Blocking



Successful of exploitation Adobe Flash vulnerability
and execute the arbitrary code without myndis Driver



What next?

Conclusion

- Future usage of myndis Driver
 - Blocking shellcode based on signature
 - Blocking Web Attack e.g. SQL Injection, Cross Site Scripting, browser exploit etc
 - Blocking ARP Poisoning Attack
 - Temporary solution for blocking exploit code as if no patches release by vendor.

Corporate Office:
CyberSecurity Malaysia,
Level 8, Block A,
Mines Waterfront Business Park,
No 3 Jalan Tasik, The Mines Resort City,
43300 Seri Kembangan,
Selangor Darul Ehsan, Malaysia.

T +603 8946 0999

F +603 8946 0888

www.cybersecurity.my

Thank You

